Parallel Cellular Programming Skeleton

Fernando Saez and Marcela Printista

LIDIC
Universidad Nacional de San Luis
Ejército de los Andes 950, San Luis, Argentina.
e-mail: {**bfsaez@unsl.edu.ar, mprinti@unsl.edu.ar**}

Abstract. Cellular automata provide an abstract model of parallel computation that can be effectively used for modeling and simulation of complex phenomena and systems. The design and implementation of parallel programming languages based on skeletons simplify the design, test and code of parallel algorithms. We discuss here the main characteristics of cellular automata programming models and we show a cellular automata skeleton for the exploitation on the inherent parallelism of problems of this kind. As a practical example, we show how to solve the problem of heat equation through the skeleton.

Keywords: Skeletal Programming, Cellular Automata, Parallelism

1 Introduction

A cellular automata (CA) is a discrete model of a system that varies in space and time. The discrete space is an array n-dimensional of identical cells, each representing a local state. As time advances in discrete steps, the system evolves according to universal laws. Every time the clock ticks, the cells update their states simultaneously. The next state of a cell depends only on the current state of the cell and its nearest neighbors.

The history of cellular automata dates back to the forties with Stanislas Ulam [5] and von Neumann [6], but it was not until the popularity of the game of Life [3] that they became more widely known. Afterwards, Stephen Wolfram developed the CA theory [7],[8].

However, today with the rapid advances in computational resources, CA have become increasingly used for more general computer simulation and modeling. The CA model can be effectively used both as a realistic approach to define abstract parallel machines and as a programming methodology for computational science on parallel computers. We are interested in this last approach.

Cellular automata are simple mathematical idealizations of natural systems. They consist of a lattice of cells (one or multi dimensional) connected with a particular geometry, and where each cell can be in one of a finite number of states.

The values of the cells evolve in discrete time steps according to deterministic rules that specify the value of each cell in terms of the values of neighboring cells and previous values.

A two-dimensional CA is a 4-upla (L, S, N, Ψ), where:

- $L = \{c(i,j) | i, j \in \aleph, 0 \leq m, 0 \leq n\}$ is a lattice mxn where $c(i,j)$ is the cell in the i-row and j-column.
- $S = \{s1, s2, ..., sr\}$ Let the set of possible states for a cell in the lattice. The state of a cell $c(i,j)$ in the time t is $s^t(i,j)$.
- $N(i,j)$ Let the neighborhood of a cell $c(i,j)$ in the time t. The neighborhood of $c(i,j)$ is composed of $s^t(i,j)$ and the values of neighboring cells.
- $\Psi(N(i,j))$ is the local rule, called $transitionfunction$, which determines the state, $s^{t+1}(i,j)$, of a cell $c(i,j)$ in the time $t+1$, based on the state of its neighbors.

The definition of n-dimensional CA is similar to that of two-dimensional CA, the lattice becomes n-dimensional and c become vector of length n.

Below we highlight the most important characteristics that define the behavior of a CA:

Initial State: The initial configuration determines the dimensions of the lattice, the geometry of the lattice, and the state of each cell at the initial stage.

States: The basic element of a CA is the cell. Each cell in the regular spatial lattice, can take any of a finite number of discrete state values. In the simplest case, each cell can have the value 0 or 1. In more complex case, the cells can have more different values. (It is even thinkable, that each cell has a complex structure with multiples values.)

Rules: The cellular automata's rules determine how the states of its cells change. They work like this: When the time comes for the cells to change state, each cell looks around and gathers information on its neighbors' states. (Exactly which cells are considered "neighbors" is also something that depends on the particular CA.) Based on its own state, its neighbors' states, and the rules, the cell decides what its new state should be. All the cells change state at the same time.

In general, if there are K states and if each cell is taken to have N neighbors (including itself), then there are K^N rules. (That's K raised to the power, or K multiplied by itself N times.)

Neighborhood: For each cell of the CA, there are a set of cells called neighborhood (usually including the cell itself) For example, the neighborhood of a cell might be defined as the set of cells with distance 3 or less.

A characteristic of a CA is that all cells have the same neighborhood structure, even the cells at the boundary of a lattice has neighboring cells that could be outside the domain. Traditionally, *border cells* are assumed to be connected to the cells on the opposite boundary as neighbors forming a closed domain. Of course, the type of boundary condition to be used depends on the application under consideration. Other types of boundary conditions may be

modeled by using preset values of the cell values for the boundary nodes or writing unique update rules for the cells at the boundary.
It is also possible to modify the determination of neighborhood. If we consider a two-dimensional automata, the most common neighborhoods are:

1. The Von Neumann neighborhood comprises the four cells (North/South/East/West) orthogonally surrounding a central cell on a two-dimensional square lattice.
2. The Moore neighborhood comprises the eight cells surrounding a central cell on a two-dimensional square lattice. It's the case of the game of life;
3. The Extended Moore neighborhood is equivalent to description of Moore neighborhood above, but neighborhood reaches over the distance of the next adjacent cells.
4. The Margolus neighborhood is a completely different approach: considers $2x2$ cells of a lattice at once. It's the type of neighborhood that is used in the simulation of gas behavior.

We can resume the characteristics that define a CA:

1. The time is discrete and advance in steps progressive.
2. They have a initial configuration or initial state. They have a lattice of cell with a particular geometry and dimension.
3. All cells in the lattice have the same neighborhood structure.
4. All cells in the lattice use the same rule, in a synchronized form, for update your state.

The rest of the paper is organized as follow, in Section 2 an overview of cellular automata are presented. In Section 3 we describe the implementation of CA skeleton and discuss its main features. Section 4 reviews the problem of Laplace's equation and proposes a solution through of CA skeleton. Finally, the conclusions are presented in Section 5.

2 Overview of Cellular Algorithms

From a computational point of view, a CA is basically a computer algorithm that is discrete in space and time and operates on a lattice of cells. The Figure 1 shows a short algorithm for resolve a two-dimensional generic CA.

```
1  AutoCel(Lattice,steps)
2  init(Lattice)
3  for k = 1 to steps
4        for i = 1 to N
5             for j = 1 to M
6                  nextState(Lattice,i,j)
```

Fig. 1. CA approach

The algorithm takes as input a lattice of NxM (two-dimensional) and initializes the lattice with some initial configuration. Then, it processes an iterative relaxation process. This process is represented in the algorithm with a iteration of *steps* steps. In each time step t, the algorithm updates each cell in the lattice. The next state of an element $s^{t+1}(i,j)$ is a function of its current state and the values of its neighbors. The relaxation process ends after *steps* iterations.

CA models offer a new methodology for modeling and simulating complex physical systems. Nevertheless, in the activity of computer simulations are very crucial requirement regards the performance of the resulting applications. CA approach proved to be a good candidates to meet this requirement since it allows to obtain parallel applications. So parallel computers represent the natural architecture where CA applications might be implemented. In fact it is possible to exploit the data parallelism intrinsic to the CA programming model coming from the possibility to execute the transition function on different sublattice due to the local nature of cell interactions.

A parallel CA can be solved efficiently by multicomputer architecture of array n-dimensional of identical processor nodes, where n is the dimension of the lattice. Each node have a lattice's part that must update. The update is synchronized and executed simultaneously by all processors. If a cell and its neighbors are in the same node, the update is easy. On the other hand, when nodes want to update the border cells, they must request the values of the neighboring cells on other nodes.

Parallel Relaxation

For each time step, every node updates its own sublattice. The next value of an interior element is a function of its current value, and the neighbor values. Every application of CA requires a different set of state transitions rules. In some applications, probabilistic state transitions require the use of a random number generator that updates a global seed variable. Parallel relaxation is not quite as easy as it sounds. When a node updates row 1 of its sublattice, it needs access to row m of the sublattice of its northern neighbor. To relax its sublattice, a node must share a single row or column with each of its four neighbors. The solution to this problem is to let two neighboring lattices overlap by one row or column vector. Before a node updates its interior elements, it exchanges a pair of vectors with each of the adjacent nodes. The overlapping vectors are kept in the boundary elements of the sublattices. If a neighboring node does not exist, a local boundary vector holds the corresponding boundary elements of the entire lattice.

3 Skeleton's Implementation

Parallelism has traditionally had the contradictory objectives in computer programming research. On one hand, parallel activity is a phenomenon that reflects the natural world; furthermore, it is often desirable as a means to achieve higher

performance. On the other hand, programmers need to detect parallel parts of a problem, to construct the processes (its activities and interactions), and to map them over real processors. Humans usually perceive these parallel activities as inherently hard.

Libraries, like PVM or MPI (low-level abstract models) give the programmer control over the decomposition task and the management of communication/synchronization among the parallel processes. Although MPI does not include explicit mapping primitives, most of its implementations have a static programming style. In this case, MPI support can avoid the mapping and scheduling problems, however, the programmer's task becomes more complex. Its unstructured programming model based on explicit, individual communications among processors is notoriously complicated and error-prone. To reduce this limitation and to increase the level of abstraction without lowering the performance, an approach exists to restrict the form in which the parallel computation can be expressed. This can be done at different abstraction levels. The model provides programming constructs: skeletons, that directly they correspond with frequent parallel patterns. The programmer expresses parallelism using a set of basic predefined forms with solution to the mapping and restructuring problems.

Following this approach, we have implemented a versatile cellular automata skeleton. The skeleton was written in C and MPI (message passing interface), and is accessed through a call to the constructor `CA_Call`. The skeleton hides the difficulties in parallel programming (e.g. data distribution, communication among processors, synchronization, etc.) and enables to write CA algorithms in easy way.

The prototype for CA skeleton is as follows:

```
void CA_Call(int dimLatt, int dimSizeLatt[], int NP,
             char *strNeighborhood, TPF_Next_State nextState,
             TypeG *Input, TypeG *Output, int SizeDataTypeCell,
             unsigned long int steps, MPI_Comm comm)
```

The number of parameters in the call to the skeleton `CA_Call` is a little large, but this long parameter list allows substantial flexibility, which will bring benefits in different application domains. The first two parameters specify the number of dimensions of the lattice (`dimLatt`) and the length for each dimension (`dimSizeLatt`). The first version of the prototype supports lattices of one and two dimensions, but is expected in future versions support to 3d lattices (This type of lattice is often used to simulate brain tumor growth, phenomena of materials, time simulations, etc). The third parameter, `NP`, is the number of processors of the multicomputer machine. Depending on the configuration of the lattice (number of dimensions and lengths) and the number of processors, the skeleton makes the best possible data partition and assigns each sublattice to a processing node. The forth parameter, `strNeighborhood`, specifies the neighborhood of a cell. The programmer may include any configuration of neighborhood in the character string. Basically the string is composed of the positions of neighbor cells in the different dimensions. The function pointer `nextState` pointing to activity responsible for change the state of a cell. Independently of the configuration used, we can consider two data structures: `Input` and `Output`. The

`Input` structure represents the input's lattice to relaxation process. The `Output` structure represents the relaxed lattice or solution of algorithm. The parameter `SizeDataTypeCell` specifies the size, in bytes, of a cell's state. It is required for skeleton to know which are the elements on `Input`. The `step` parameter specifies the number of iterations on CA.

Then we develop how are configured each of the main aspects of a CA in the skeleton.

3.1 Initial State

The initial state determines the dimensions of the lattice (e.g., one, two and three dimensions), the geometry of the lattice, and the state of each cell at the initial stage. In the skeleton these aspects are solved by setting the lattice's dimensions and its lengths. With reference to geometry, the skeleton only accept rectangular lattice (cell's have two neighbor in each dimension). It divides the lattice and assigns each sublattice to a processor.

The initial values (states) must be fill for the programmer into the `Input` structure.

3.2 States

The basic element of a CA is the cell. Classic CA problems include cells with binary states and cells with a range of integer values, but complex CAs include cells with real values, and even the state of a cell could be as complex as a structure or other non-simple type.

In this sense, the skeleton provides a data-type, `TypeG`, that allows a programmer to configure any data-type to a cell. For this `CA_Call` requires a parameter `SizeDataTypeCell` with the size of bytes of a cell.

The CA skeleton include the implementation of the `TypeG` data-type and the operands for work with it (alloc memory, copy's functions, etc). Of course, is responsibility of programmer make the cast necessary to a correct program.

3.3 Rules

The cellular automata's rules determine how the cell's states change during the relaxation. In the diversity of cellular automata there are hundreds of rules that define the behavior of the AC. For example Stephen Wolfram [7] focused its attention on particular simple one dimensional CAs. The CA has two possible states for each cell, and a neighborhood consisting of the sites to the immediate right and left of a given cell. There are 8 possible configurations for such neighborhoods, and so 256 possible elementary CA rules; in the paper, Wolfram introduced an useful scheme for referring to those rules, and others, by number, so that we speak of rule 18, rule 22, rule 90, rule 110, etc. Beyond that, while the rules were simple, the patterns they could generate were complicated. Other set of rules was defined for changing the state of a cell in the classic game of life [3].

No matter what the problem is attacked, a cell changes its current value through a set of rules that define the next state of the cell depending on its current value and the value of its neighboring cells.

The rules are described by functions (`nextState`) and they will need to be implemented by the programmer. The parameters of function define the cell to update and the lattice with your neighboring cells. The value assigned to a cell in the current interaction will be really available only in the next iteration.

3.4 Neighborhood

In cellular automata we compute the value of each cell in the next generation by performing a local computation based on the cell and the neighborhood of that cell. In our search for a flexible CA template that fits a lot of problems, we must allow the programmer to specify the neighborhood used in a versatile and easy manner. This is achieved through the neighborhood string `strNeighborhood`.

The string specifies the cells list using a character semicolon to separate each cell. Then, a cell is represented by a tuple where each element is an offset, for some dimension, relative to the central cell.

For example, the programmer can specify the classical von Neumann neighborhood to 2-dimensions as follows "-1,0;0,1;1,0;0,-1". Here the string has four cells. The first cell, (-1,0), represents the left neighbor, the second cell, (0,1), represents the upper cell, third and forth cell represent the right and lower cell. Also, the programmer can specify a Moore neighborhood adding more cells to von Neumann neighborhood like "-1,0;0,1;1,0;0,-1;-1,1;1,1;-1,-1;1,-1"

3.5 Relaxation

Relaxation is the main process of any CA. It represents the progress of time in discrete steps necessary to solve a problem or perform a simulation. In our implementation, each processing node has a sublattice and an additional column of border cells for each sublattice side. These border cells represent the connected cells located on the adjacent processors.

At every iteration, a processing node computes the next state for its cells, and then exchanges its additional columns with its neighbors. These border cells are used for the next iteration.

The process of relaxation is repeated as necessary. In the skeleton the programmer set this option through parameter `steps`.

4 A simple example

Cellular automata applications are diverse and numerous:

- Simulation of gas behavior. A gas is composed of a set of molecules whose behavior depends on the one of neighboring molecules.
- Simulation of percolation process.

– Simulation of forest fire propagation.
– Simulation and study of urban development.
– Simulation of crystallisation process.

In a different field, cellular automata can be used as an alternative to differential equations. The numerical solution of, say, Laplace's equation by lattice relaxation is really a discrete simulation of heat flow performed by a cellular automata [1][4][2].

Heat Equation

Our example is based on obtaining the laplace's equation for equilibrium temperatures in a square region of homogeneous material, uniform thickness h and with fixed temperatures at the boundaries.

A numerical solution to the heat equation is based on the observation that the heat flow in elements of the interior is driven by differences in temperatures between the elements and their neighbors (Figure 2).

Fig. 2. Neighbors elements

On equilibrium temperatures each one of interior element can be calculated as the average of the temperatures of its four neighbors.

$$u_c \simeq (u_n + u_s + u_e + u_w)/4 \tag{1}$$

4.1 Implementation

We show, in the Figure 3, an easy and efficient algorithm, using the CA skeleton, that solve the heat equation for a small square region (divided into a lattice of NxN elements), where boundaries of the region, have a known fixed temperature. This configuration is performed at the beginning of the algorithm.

The algorithm configures a two-dimension lattice of NxN cells, where each cell represents a temperature. The value `NP` set the number of processors of the multicomputer. The variable `VonNeumann` has the neighbors string that represents the classical Von Neumann neighborhood. The `nextState` function is responsible for update the temperature. It changes the value of the cell in function of the Equation 1. The last parameters set the size, in bytes, of the cell, the total of steps in relaxation and the communicator of MPI. At the ending of

```
int SizeLatt[2] = {N,N};
TypeG *Input, Output;
initL(Input);
initL(Output);
char VonNeumann[10] = "-1,0;0,1;1,0;0,-1";
initializeConfiguration(Input);
CA_Call(2, SizeLatt, NP, VonNeumann, &nextState, Input, Output, sizeof(double),steps,comm)
            \\master processor assembles a complete lattice ..
```

Fig. 3. Algorithm for Laplace's equation

the algorithm a master processor gathers each `Output` and assembles a complete lattice

This algorithm that has been coded in a program composed of a few lines of codes would requiere several hundred lines of code if a languaje without a cellular automata skeleton was used.

4.2 Results

We performed initial experiments. Table 1 shows the execution time for differents configurations over cluster $LIDIC$. The cluster consist of 14 networked nodes, each one a Pentium IV of 3.2 GHz. The nodes are connected together by Ethernet segments and a Switch Linksys srw 2024 of 1 GB. We ran the experiments with four and nine processing nodes. For these configurations the skeleton ensures a architecture type grid that is ideal for the solution of the square matrices. The presented results closely match the expected results.

Table 1. Sequential and Parallel Times.

N	T_{Sec}	$P=4$	Sp	$P=9$	Sp
100	0.075009	0.147076	0.51	0.182949	0.41
200	0.286344	0.298275	0.96	0.325391	0.88
500	0.754242	0.496212	1.52	0.474366	1.59
1000	1.508485	0.704899	2.14	0.661616	2.28
2000	3.016969	1.323232	2.28	0.877026	3.44
4000	6.033939	2.567634	2.35	1.169368	5.16

5 Conclusions and future work

We have presented the main characteristics that define the behavior of a cellular automata. The initial state, the values of the cells in the lattice, the rules, the neighborhood, and the relaxation process are the characteristics that are combined and allow to CA to reproduce different patterns.

This approach saw from a computational point of view is basically a computer algorithm that is discrete in space and time, and that performs the same

operation for all cells on a lattice. This aspect made of the algorithm an ideal candidate for data parallelism.

In this sense, further considerations arise. The classical problems inherent to parallel programming like the decomposition task, the decomposition data, mapping and scheduling of task over real processors, the management of communication/synchronization, load balance and others make more difficult the task of programming by the programmer.

To reduce this complexity and to increase the level of abstraction we proposed to use the CA skeleton. This skeleton provides a procedure `CA_Call` that can be add to a MPI program in an easy form, and allow to programmer to focus in application's aspects.

The programmer tasks are: initialize the input lattice, set the procedure's parameters and developt the function `nextState`.

The Preliminary results show a expected speedup, however we are currently conducting a series of experiments that allow a better analysis of the skeleton.

At present we continue to work in parallel cellular automata, studying support to advance characteristics (load balance, nested parallelism, etc). The portability and optimization of the skeletons over differents architectures (both shared and distributed memory) are important issues for a skeletal programming systems that will bring its rewards.

Acknowledgments

We wish to thank the Universidad Nacional de San Luis, the ANPCYT and the CONICET from which we receive continuous support.

Bibliografa

1. R.H. Barlow and D.J. Evans. *Parallel algorithms for the iterative solution to linear systems*. Computer Journal 25, 1, 56-60., 1982.
2. Johnson M.A. Lyzenga G.A. Otto S.W. Salmon J.K. Fox, G.C. and D.W. Walker. *Solving Problems on Concurrent Processors*. Vol. I, Prentice-Hall, Englewood Cliś, NJ., 1988.
3. Martin Gardner. *The Fantastic Combinations of John Conwayś New Solitaire Game of Life*. Scientific American. 120-123, 1970.
4. Askew C.R. Carpenter D.D. Glendinning I. Hey A.J.G. Pritchard, D.J. and D.A. Nicole. *Practical parallelism using transputer arrays*. Lecture Notes in Computer Science 258, 278-294, 1987.
5. S. Ulam. *Some ideas and prospects in biomathematics*. Anm.Rev.Biophys.Bioengin. 1 : 277-291, 1963.
6. J. von Neumann. *Theory of Self-Reproducing Automata (edited and completed by Arthur Burks)*. University of Illinois Press, 1966.
7. S. Wolfram. *Statistical mechanics of cellular automata*. Rev. Mod. Phys. 55: 601, 1983.
8. S. Wolfram. *Computation theory of cellular automata*. Commun. Math. Phys. 96 15-57, 1984.